Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 83 (2016) 1182 – 1187
Procedia Computer Science

The 6th International Symposium on Internet of Ubiquitous and Pervasive Things (IUPT 2016)

Linking the Web of Things: LDP-CoAP mapping

Giuseppe Loseto, Saverio Ieva, Filippo Gramegna, Michele Ruta*, Floriano Scioscia, Eugenio Di Sciascio

Politecnico di Bari, via E. Orabona 4, Bari (I-70123), Italy

Abstract

The Linked Data Platform (LDP) W3C Recommendation defined resource management primitives for HTTP only, pushing into the background not-negligible use cases related to Web of Things (WoT) scenarios where HTTP-based communication and infrastructures are unfeasible. This paper proposes a mapping of the LDP specification for Constrained Application Protocol (CoAP) in order to publish Linked Data on the WoT. A general translation of LDP-HTTP requests and responses is provided, as well as a fully comprehensive framework for HTTP-to-CoAP proxying. The theoretical work is corroborated by an experimental campaign using the W3C Test Suite for LDP.

Peer-review under responsibility of the Conference Program Chairs
Keywords: Linked Data Platform, CoAP, Semantic Web of Things

1. Motivation

The World Wide Web Consortium (W3C) has recently released the Linked Data Platform (LDP) specification[1]. It aims to provide a reference format for exposing and managing LD resources on the Web. Particularly, clear and direct guidelines are now given for resource classification made according to resource type. Although this standardization effort improves previous RDF graphs management based on SPARQL 1.1 Graph Store HTTP protocol (https://www.w3.org/TR/sparql11-http-rdf-update/) and basically fixes multiple issues, it leaves out Web of Things (WoT) scenarios where alternative lightweight application protocols surrogate HTTP. This is, for example, the case of CoAP (Constrained Application Protocol)[2], a level 7 standard designed for Machine-to-Machine (M2M) communication of constrained devices in the Internet of Things. Following the REST (REpresentational State Transfer) architectural style, CoAP adopts a loosely coupled client/server model, based on stateless operations on *resource* representations[3]. Each resource is unambiguously identified by a URI (Uniform Resource Identifier). Clients access resources via asynchronous request/response interactions over a datagram-oriented transport like UDP, using HTTP-derived methods essentially mapping the Read, Create, Update and Delete operations of data management.

Hence, main motivation of the paper stems from the need of extending and enriching the standardization of Linked Data Platforms also to Web of Things use cases. It should be said that the W3C indicated possible solutions for

* Corresponding author. Tel.: +39-339-635-4949; fax: +39-080-596-3410
E-mail address: michele.ruta@poliba.it

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2016.04.244

resource management in the WoT (see sec. 3.12 of Linked Data Platform Use Cases and Requirements[4]), but their scope is quite limited. In fact, an approach based on a one-to-one HTTP-CoAP translation was followed[5]. Unfortunately, such a mapping only worked with basic HTTP interactions, where several methods and headers were not used and/or some other ones were particularly simplistic. As an example the methods *options*, *head* and *patch* were not allowed as well as several MIME types (content-format) were missing, so that not negligible constraints on resources introduced by LDP could not be considered. Hence, by trivially applying the one-to-one mapping suggested by the W3C for coping with WoT scenarios, significant LDP functionality would be lost.

This paper proposes a novel and specific variant of the HTTP-CoAP mapping able to preserve all the LDP features and capabilities in the Web of Things with a full support of the W3C specification. It also adds modern features giving value to the strongest potentialities of CoAP (*e.g.*, resource discovery based on CoRE Link Format), which are completely absent in HTTP.

The remainder of the paper is organized as follows. The next section presents some background about LDP and CoAP in order to make the work self-consistent, Section 3 introduces the proposed LDP-CoAP mapping, while Section 4 addresses early experiments made to validate and corroborate the proposal. Finally, Section 5 closes the paper.

2. Background

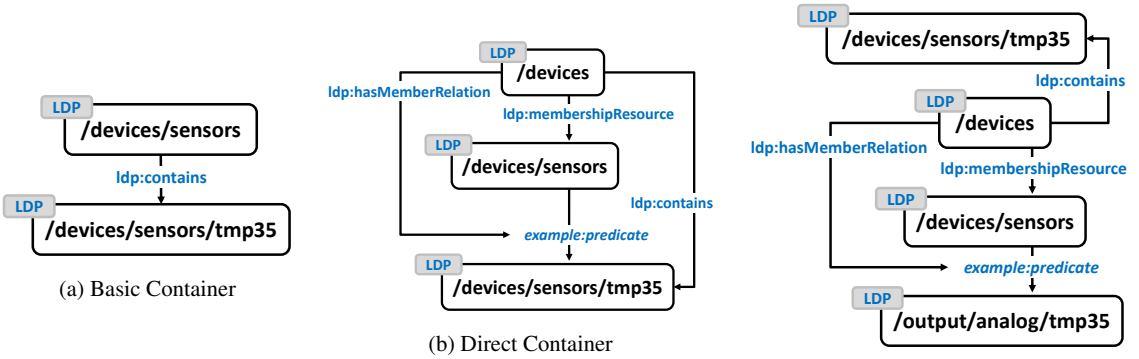
2.1. Linked Data Platform

LDP W3C Recommendation states *Linked Data Platform defines a set of rules for HTTP operations on web resources, some based on RDF, to provide an architecture for read-write Linked Data on the web*. In particular, the specification describes the use of HTTP methods and headers for accessing and managing resources from *LDP servers* following the Linked Data approach[6]. Basically, seven types of Linked Data Platform Resources (LDPRs) are defined, conforming to simple patterns and conventions.

LDP Resource (LDPR): a HTTP resource whose status complies with the basic LDP guidelines;
LDP RDF Source (LDP-RS): a LDPR whose status corresponds to an RDF graph and can be fully represented in a RDF syntax. In particular LDP allows `text/turtle`[7] and `application/ld+json`[8] serializations;
LDP Non-RDF Source (LDP-NR): a LDPR not represented in RDF, *i.e.*, a binary or text document without useful RDF annotation. LDP servers can also generate metadata about LDP-NR resources, *e.g.*, creation date or owner;
LDP Container (LDPC): a LDP-RS as collection of LDP resources. Three types of LDPC are defined, namely *Basic*, *Direct* and *Indirect*;
LDP Basic Container (LDP-BC) is a LDPC defining a simple link to its resources through the `ldp:contains` predicate, as shown in Figure 1(a);
LDP Direct Container (LDP-DC) is a LDPC increasing the flexibility of a LDP-BC with the *membership* feature. A LDP-DC contains membership triples, according to the pattern in Figure 1(b), specifying the *membership resource* and the *member relation*.
LDP Indirect Container (LDP-IC) is a LDPC similar to a LDP-DC, also capable of having member resources with different URIs, as shown in Figure 1(c), unrelated to the main container URI. These resources are specified using the *insertedContentRelation* LDP property in the body of LDP requests.

LDP specification also defines required and optional HTTP methods for LDP servers:
- **GET**: retrieves the description associated to the selected LDP resource;
- **POST**: creates a LDP resource in a LDP container;
- **PUT**: creates or updates a LDP resource in a LDP container;
- **DELETE**: removes a LDP resource on a LDP server;
- **HEAD**: retrieves the same HTTP headers as GET responses without body content;
- **OPTIONS**: lists the operations allowed on a LDP resource by means of specific HTTP response headers;
- **PATCH**: allows LDP clients to update a resource description exploiting the *Linked Data Patch Format*[9].

The *W3C LDP Implementations* reference page (*http://www.w3.org/wiki/LDP_Implementations*) enumerates several frameworks proposed in the last years. The most relevant ones are summarized in Table 1, showing both main features



(a) Basic Container

(b) Direct Container

(c) Indirect Container

Fig. 1: Membership patterns of LDP containers (`ldp` prefix used as abbreviation of the LDP namespace `http://www.w3.org/ns/ldp#`)

and supported resources, as reported in the *LDP implementation report*[10]. All implementations are based on the HTTP protocol. No support is currently given to WoT standards, such as CoAP.

Name	Status	Last Version	License	Language	Supported LDP Resources
Apache Marmotta	Full release	May 2014	APL 2.0	Java	RS, NR, BC
Eclipse Lyo	Completed	Aug 2014	EPL 1.0	Java	RS, NR, BC, DC
rww.io	Pending	Nov 2014	MIT	PHP	RS, BC
LDP.js	Completed	Apr 2015	APL 2.0	JavaScript	RS, BC
Fedora 4.4	Full release	Oct 2015	APL 2.0	Java	RS, NR, BC, DC, IC
Carbon LDP	In progress	Oct 2015	BSD	JavaScript	RS, NR, BC, DC, IC
LDP4j	In progress	Dec 2015	APL 2.0	Java	RS, BC, DC, IC
rww-play	In progress	Dec 2015	APL 2.0	Scala	RS, NR, BC
OpenLink Virtuoso	Full release	Dec 2015	GPLv2	C/C++	RS, BC
gold	In progress	Jan 2016	MIT	Go	RS, BC
Callimachus	Full release	Jan 2016	APL 2.0	Java	RS, NR, IC

Table 1: Current LDP implementations

2.2. Constrained Application Protocol

A CoAP message is composed of: (i) a 32-bit *header*, containing the request method code (or response status); (ii) an optional *token* value, used to associate replies to requests, (iii) a sequence of *option* fields (containing information such as resource URI and payload media type), (iv) the *payload* data. The CoRE Link Format specification[11] is adopted for resource discovery. A client accesses the reserved URI `/.well-known/core` on the server via `GET` to retrieve available resource entry points. Further `GET` requests will include `URI-query` options to retrieve only resources with given attributes. Standardized query attributes include resource type (`rt`), interface usage (`if`), content-type (`ct`), and MIME (Multipurpose Internet Mail Extension) `type` for a resource. Further non-reserved attributes can be freely used. CoAP also supports proxying, enabling Web applications to transparently access the resources hosted in devices based on CoAP. A CoAP proxy can be explicitly selected by clients (forward-proxy) or can act as the origin server for the target resource (reverse-proxy). Particularly, HTTP-CoAP cross-protocol proxying allows HTTP clients to access resources on CoAP servers; a HTTP request containing a Request-URI with *coap* or *coaps* scheme is forwarded to the specified CoAP resource.

3. LDP-CoAP mapping

A novel HTTP to CoAP mapping tought for LDP is presented here, improving the proposal in[5]. LDP-HTTP request methods and headers have been properly translated to the corresponding LDP-CoAP ones and LDP-CoAP

responses are then mapped back to LDP-HTTP. The proposed mapping enables direct CoAP-to-CoAP interaction among devices supporting LDP-CoAP.

Basically, the proposed LDP-CoAP association is obtained applying the following rules:

HTTP methods (shown in Table 2) are translated to the corresponding CoAP methods (if present). `PATCH`, `HEAD` and `OPTIONS`, not defined in CoAP, are mapped to existing methods, adding the new Core Link Format attribute `ldp`. This solution extends the basic CoAP functionalities while maintaining a full compatibility with the standard protocol.

HTTP status codes are mapped with available CoAP codes as described in Table 2. Codes for bad requests and errors are translated as defined in the proposal[5].

HTTP Headers of request/response messages are translated as in Table 3.

Finally, novel **content-format** media types are introduced in CoAP: `text/turtle` and `application/ld+json`.

HTTP Method	Mandatory	Supported in CoAP	LDP-CoAP	HTTP SC	CoAP SC
GET	YES	YES	GET	200 OK	2.05 Content
POST	NO (optional)	YES	POST	201 Created	2.01 Created
PUT	NO (optional)	YES	PUT	204 No Content	2.04 Changed
DELETE	NO (optional)	YES	DELETE	204 No Content	2.02 Deleted
PATCH	NO (optional)	NO	PUT ?ldp=patch	204 No Content	2.04 Changed
HEAD	YES	NO	GET ?ldp=head	204 No Content	2.03 Valid
OPTIONS	YES	NO	GET ?ldp=options	204 No Content	2.05 Content

Table 2: HTTP-CoAP Methods and Status Codes (SCs) Mapping

HTTP Header	LDP-CoAP
Content-Type	Content-Format (`ct`) CoAP option
Link (`rel="type"`)	Resource-Type (`rt`) Core Link Format attribute, available through a CoAP discovery request
Allow Accept-Post Accept-Patch	Not defined in CoAP, available in JSON format as body content of a LDP-CoAP Options request
Slug	`title` Core Link Format attribute
Location	`location-path` CoAP option

Table 3: HTTP-CoAP Headers Mapping

In order to make everything clearer, in what follows some reference examples of HTTP-CoAP translation are given. Note that in some cases an HTTP request cannot be translated into a single CoAP request, but more CoAP messages are needed.

Example 1. Basic HTTP GET request on an LDP resource

```
GET /alice/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

The HTTP response is shown in Figure 2. In this case, a single CoAP GET request is not able to produce all the required headers, because some of them are not defined in the response format of CoAP. So the original HTTP request is translated to the following three LDP-CoAP requests:

- a `GET` message to map Content-Type (`ct`), ETag (if present) and RDF content of the LDP resource;
- a CoAP discovery message to retrieve the `rt` attribute indicating the LDP type of each resource. It maps the HTTP *Link* response header;
- an `OPTIONS` message (described later) to map the *Allow*, *Accept-Post* and *Accept-Patch* response headers.

Example 2. Create a new LDP resource through a HTTP POST request

```
POST /alice/ HTTP/1.1
Host: example.org
Slug: foaf
Content-Type: text/turtle
<payload>
```

In this case, the request is translated to a single CoAP `POST` message with URL:

```
coap://example.org/alice?title=foaf&rt=ldp:Resource ct=text/turtle <payload>
```

```
HTTP/1.1 200 OK
Content-Type: text/turtle; charset=UTF-8
Link:       <http://www.w3.org/ns/ldp#BasicContainer>; rel="type",
            <http://www.w3.org/ns/ldp#Resource>; rel="type"

Allow: OPTIONS,HEAD,GET,POST,PUT,PATCH
Accept-Post: text/turtle, application/ld+json, image/bmp, image/jpeg
Accept-Patch: text/ldpatch
Content-Length: 250
ETag: W/'123456789'
```

DISCOVERY REQUEST

OPTIONS REQUEST

Fig. 2: Example1. HTTP GET response (payload data not included)

As defined in Table 3, `title` and `rt` query parameters are obtained from the *Slug* and *Link* HTTP header fields, respectively. If the *Link* header is not defined, `ldp:Resource` is used as default value of `rt`. The HTTP response will contain the *Location* HTTP header corresponding to the *Location-Path* CoAP response option.

Example 3. HTTP OPTIONS request on a LDP resource

An `OPTIONS` request is used to obtain useful information about a resource, *e.g.*, the list of available methods. HTTP `OPTIONS` response is shown in Figure 3. Also in this case, multiple LDP-CoAP requests are combined to obtain all the headers produced by the HTTP reply:

- *Allow*, *Accept-Post* and *Accept-Patch* response headers are not defined in CoAP so their values are set in the LDP-CoAP `OPTIONS` response body in JSON syntax and then mapped to the corresponding HTTP headers;
- a CoAP discovery request is used to obtain the resource type (`rt`) then mapped to the HTTP *Link* response header.

Further examples for each method are on our LDP-CoAP Web page (*http://sisinflab.poliba.it/swottools/ldp-coap*).

```
HTTP/1.1 204 No Content
Allow: OPTIONS,HEAD,GET,POST,PUT,PATCH
Accept-Post: text/turtle, application/ld+json, image/bmp, image/jpeg
Accept-Patch: text/ldpatch
Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type",
<http://www.w3.org/ns/ldp#Resource>; rel="type"
```

DISCOVERY REQUEST

Fig. 3: Example 3. HTTP OPTIONS response

4. Validation

The validation framework consists of four elements, shown in Figure 4:

- *LDP-CoAP Server*, a CoAP server exposing resources complying with LDP-CoAP;
- *CoAP Client*, making requests to the LDP-CoAP server through CoAP;
- *HTTP Client*, querying through HTTP messages a web server which exposes LDP resources. It does not communicate directly with a LDP-CoAP server;
- *LDP-CoAP Proxy*, an HTTP-to-CoAP device used to connect CoAP devices to HTTP-based networks. It is responsible for: (i) processing requests from HTTP clients; (ii) mapping HTTP requests to compatible LDP-CoAP ones via the mapping rules described in Section 3; (iii) forwarding requests to the LDP-CoAP server; (iv) translating the LDP-CoAP responses to HTTP responses to be returned to the client.

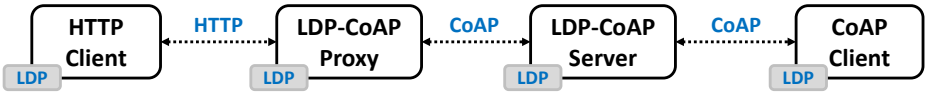


Fig. 4: LDP-CoAP validation architecture

Californium Java library [12] was used to implement the LDP-CoAP framework. In particular, the LDP-CoAP proxy was based on the *californium-proxy* package, whilst the LDP-CoAP server exploited *californium-core* and *OpenRDF Sesame 2.8.6* library (`http://rdf4j.org/`) for RDF data processing and storage.

Functionality of the proposed framework was evaluated through the W3C LDP Test Suite (*http://w3c.github.io/ldp-testsuite/*). By default, the suite diyurectly queries a LDP server by means of HTTP messages; therefore for LDP-CoAP tests requests were sent to the server through a LDP-CoAP proxy as in Figure 4. The suite consists of 236 tests referred to rules and restrictions of the LDP W3C specification. Table 4 reports on the obtained results grouped by supported LDP resources: RDF Sources, Non-RDF Sources and Basic, Direct, Indirect Containers. For each test category, the specification requirements are divided in three compliance levels: MUST, SHOULD, and MAY. Due to the lack of space, extensive reports with full details are not reported here, but can be found on the project Web page. Currently unsatisfied test cases are related to the following unsupported features: PUT-to-create and PATCH methods, paging and sorting, preference HTTP headers.

Feature	MUST	SHOULD	MAY
Basic Container	32/37 (86.5%)	12/17 (70.6%)	3/4 (75.0%)
Direct Container	37/42 (88.1%)	13/19 (68.4%)	3/4 (75.0%)
Indirect Container	33/39 (84.6%)	12/17 (70.6%)	3/4 (75.0%)
Non-RDF Source	12/15 (80.0%)	1/1 (100.0%)	4/6 (66.7%)
RDF Source	22/24 (91.7%)	5/7 (71.4%)	1/1 (100.0%)

Table 4: LDP-CoAP Test Suite results summary

5. Conclusion and Future Work

This paper introduced a CoAP mapping of the Linked Data Platform specification for publishing Linked Data on the Web of Things. The LDP W3C Recommendation, which defined resource management primitives only for HTTP, stated the need for this work. The proposal includes a translation of LDP-HTTP requests and responses, as well as a framework for HTTP-to-CoAP proxying.

Future work will include a performance evaluation of the proposed framework to assess the impact of LDP support in resource-constrained devices. Furthermore, mapping the currently unsupported LDP features is under way, in order to increase the compatibility with the specification: running the test suite again on future revisions will measure progress in this area. Finally, a complete scenario will be defined to expose both real-time data and sensor observations (*e.g.*, weather data[13]) according to LDP-CoAP specifications.

References

1. A. Malhotra, J. Arwe, S. Speicher, Linked Data Platform 1.0, W3C Recommendation, W3C, http://www.w3.org/TR/ldp/ (Feb. 2015).
2. Z. Shelby, K. Hartke, C. Bormann, The Constrained Application Protocol (CoAP), RFC 7252 (Proposed Standard) (Jun. 2014). URL `http://www.ietf.org/rfc/rfc7252.txt`
3. C. Bormann, A. Castellani, Z. Shelby, CoAP: An Application Protocol for Billions of Tiny Internet Nodes, IEEE Internet Computing 16 (2) (2012) 62–67.
4. S. Battle, S. Speicher, Linked Data Platform Use Cases and Requirements, W3C Working Group Note, W3C, http://www.w3.org/TR/ldp-ucr/ (Mar. 2014).
5. A. Castellani, S. Loreto, A. Rahman, T. Fossati, E. Dijk, Guidelines for HTTP-CoAP Mapping Implementations, Internet-Draft draft-ietf-core-http-mapping-07, IETF Secretariat (July 2015).
6. T. Heath, C. Bizer, Linked data: Evolving the web into a global data space, Synthesis lectures on the semantic web: theory and technology 1 (1) (2011) 1–136.
7. G. Carothers, E. Prud'hommeaux, RDF 1.1 Turtle (Terse RDF Triple Language), W3C Recommendation, W3C, http://www.w3.org/TR/turtle/ (Feb. 2014).
8. M. Lanthaler, M. Sporny, G. Kellogg, JSON-LD 1.0 (A JSON-based Serialization for Linked Data), W3C Recommendation, W3C, http://www.w3.org/TR/json-ld/ (Jan. 2014).
9. A. Bertails, P.-A. Champin, A. Sambra, Linked Data Patch Format, W3C Working Group Note, W3C, http://www.w3.org/TR/ldpatch/ (Jul. 2015).
10. S. Speicher, S. Fernández, Linked Data Platform Implementation Conformance Report, W3C Working Group Note, W3C, http://www.w3.org/TR/ldp-implreport/ (Dec. 2014).
11. Z. Shelby, Constrained RESTful Environments (CoRE) Link Format, RFC 6690 (Proposed Standard) (Aug. 2012). URL `http://www.ietf.org/rfc/rfc6690.txt`
12. M. Kovatsch, M. Lanter, Z. Shelby, Californium: Scalable cloud services for the internet of things with coap, in: Internet of Things (IOT), 2014 International Conference on the, IEEE, 2014, pp. 1–6.
13. H. Patni, S. S. Sahoo, C. Henson, A. Sheth, Provenance Aware Linked Sensor Data, in: Proceedings of the Second Workshop on Trust and Privacy on the Social and Semantic, 2010.